A Proposed Group Management Scheme for XTP Multicast

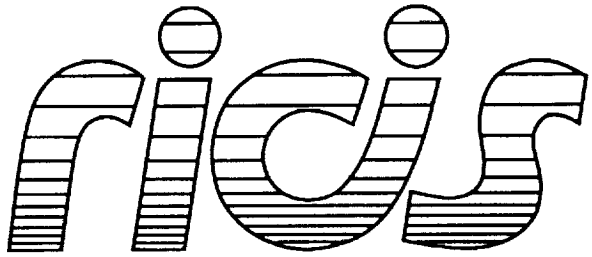
Bert J. Dempsey
Alfred C. Weaver

Digital Technology

August, 1990

Cooperative Agreement NCC 9-16
Research Activity No. SE.31

NASA Johnson Space Center
Engineering Directorate
Flight Data Systems Division



Research Institute for Computing and Information Systems
University of Houston-Clear Lake

INTERIM REPORT

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Bert J. Dempsey and Alfred C. Weaver of Digital Technology. Dr. George Collins, Associate Professor of Computer Systems Design, served as RICIS research coordinator.

Funding was provided by the Engineering Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA research coordinator for this activity was Frank W. Miller of the Systems Development Branch, Flight Data Systems Division, Engineering Directorate, NASA/JSC.

A Proposed Group Management Scheme for XTP Multicast

Bert J. Dempsey, Alfred C. Weaver

Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, Virginia 22903
(804) 924-7605
bjd7p@virginia.edu, weaver@virginia.edu

1. Definition and Purpose

Group communication involves a set of distributed processing entities, a *process group*, that corresponds to a set of transfer layer entities, each listening on the same address. This set of communication endpoints constitutes the *multicast group*. In this document the term 'groups' refers to multicast groups.

Reliability issues in one-to-many communication are more complex than with unicast. Multicast addresses identify a logical multicast group that is dynamically bound to a changeable set of receivers, some or all of which may be unknown at the multicast source. Certain applications exploit this powerful distributed binding facility to support transparent, robust service from a set of identical servers. In general, however, applications that will use transfer layer multicast will demand a wide range of multicast semantics. Process groups managing replicated data objects often can not recover easily from partial updates; also, many such groups use transaction serialization techniques such as two-phase commit protocols for concurrency control. These applications need an underlying multicast facility in which an explicit group view can be associated with each multicast transfer. Applications involving voting algorithms, in contrast, are typically indifferent to individual process group members as long as a certain number of group members are reached.

The purpose of a *group management* scheme is to enable its associated transfer layer protocol to be responsive to user-determined reliability requirements for multicasting. Group management (GM) must assist the client process in coordinating multicast group membership, allow the user to express the subset of the multicast group that a particular multicast distribution must reach in order to be successful (reliable), and provide the transfer layer protocol with the group membership information necessary to guarantee delivery to this subset. Group management provides services and mechanisms that respond to the need of the client process or

process level management protocols to coordinate, modify, and determine attributes of the multicast group, especially membership. XTP GM provides a link between process groups and their multicast groups by maintaining a group membership database that identifies members in a name space understood by the underlying transfer layer protocol. Other attributes of the multicast group useful to both the client process and the data transfer protocol may be stored in the database. Examples include the relative dispersion, most recent update time, and default delivery parameters of a group.

1.1. XTP Group Management Functionality

Our proposed XTP Group Management scheme sets up a GM entity per XTP instance. The GM entity serves as a local monitor for multicast management. Except in low probability scenarios, adding or deleting a member from an existing multicast group requires no communication over the network. This design keeps down network overhead and avoids imposing any join latency on new members. Join latency may occur due to process level group coordination. A XTP GM name server (or name server group) controls distributed naming, but the distributed nature of the XTP GM scheme design prevents failures that render the name server unreachable from, in most cases, disabling normal operation of XTP GM and, by extension, of XTP reliable multicasting.

The functionality of XTP GM may be classified into four parts. First, it will maintain a database containing multicast group status information on all groups known to the local GM entity. This database will cache group information learned from user input and as a byproduct of group communication (from XTP). The GM database represents a mechanism within the communication subsystem for supporting application-specific process group management. Second, a GM entity will respond to queries from remote GM entities. These queries may be purely internal to the GM scheme or may be prompted by user queries to the GM database that

could not be resolved locally. Third, GM provides the user with primitives for modifying a multicast group. With the correct access code, a client process can create, join, or leave a multicast group. Fourth, the client process submits reliable multicast transfer requests to XTP GM, which initializes the appropriate XTP state and supplies XTP with a data structure, a *transfer list*, to identify the number and the identifiers of required participants in the multicast exchange.

1.2. Underlying Assumptions

Our depiction of XTP GM is shaped by the assumption that multicast management will be most vital, at least initially, within single LAN environments for multicast groups with a small number of members (e.g. less than 32). Groups will have memberships that change slowly, and most multicast groups will have their group identifiers assigned at system initialization. The latter does not imply that membership will not vary, only that the binding of a logical set of multicast receivers to a group identifier (group creation) will be most frequently done at system initialization. It should be emphasized that the prejudice here toward small, slowly changing groups does not represent some fixed boundary for either set size or set modification frequency, beyond which XTP GM collapses. XTP GM will remain stable with cost as measured by consumption of network resources and multicast message latency being proportional to the product of the number of groups, their average cardinality, and their average number of membership modifications.

2. XTP Group Management Service Interface

Below we discuss the interface between the application process and XTP GM. The list to the right of the $\rightarrow$ indicate the return values of each primitive.

Create(*type*) → {*result, XTP_group_id, access*}

This primitive allows a client process to create a multicast group. The parameter *type* indicates whether modifications, queries, and transmissions involving the group will require a GM-generated access key or not. These key values

are placed in the return parameter, *access*. Unprotected groups will have null access keys. The local XTP GM entity assigns the new group an identifier associates the identifier with MAC and transfer layer multicast addresses, and returns this value to the user in *XTP_group_id*. *Result* indicates whether the call to Create() was successful.

Each XTP GM can allocate a block of identifiers in the group identifier name space (i.e. *XTP_group_ids*) by contacting the XTP GM name server. If the name server does not respond, the local XTP GM broadcasts to the well-known address for XTP GM entities and requests an unassigned *XTP_group_id*. In most cases an identifier can be assigned without network communication.

Each *XTP_group_id* has the group addresses that define the multicast group embedded within it. The exact mapping of *XTP_group_id* to transfer layer group address will be dependent on the underlying address structure. The mapping must have the following properties:

(1) it must guarantee that multiple *XTP_group_ids* are never associated with the same address.

(2) it should produce *XTP_group_ids* that have some property that allow them to be checked for validity when passed to XTP GM.

(3) it should be invertible; that is, XTP GM should be able to recover the address from the group identifier while the user should not.

The multicast group address includes a medium access control (MAC) layer group address. Host interfaces to the network tend to support a relatively small number of MAC group filters. Efficient use of filters is important to protect hosts from multicast traffic that is superfluous for them. Hence XTP GM should ensure that hardware filters are selected at random using a uniform distribution over the range of all supported filters in order to optimize their effectiveness over the entire LAN.

Join(*XTP_group_id, access*) → {*result, XTP_member_id*}

If *access* is valid, the join() primitive causes XTP GM to set up an XTP context that listens on the address associated with *XTP_group_id*. The XTP context is given a GM-generated identifier, unique within the group *XTP_group_id*, and this identifier is returned to the client process in *XTP_member_id*. The local XTP GM database is updated to reflect the new member.

Blocks of *XTP_member_ids* will also be allocated by the XTP GM name server. If the server does not respond, member identifiers can be solicited from remote XTP GM entities. These identifiers are reused without a waiting period after deallocation by the Leave() group primitive. Unlike group identifiers, *XTP_member_ids* must be small (e.g. 4 bytes) since XTP will also use this identifier for identifying the group member during multicast transfers.

Leave(*XTP_group_id, XTP_member_id, access*) → {*result*}

XTP GM shuts down the XTP context associated with the specified group member, deallocates the member identifier for local reuse, and updates the local group membership database.

In addition to the explicit Leave() primitive, leaving a group may be an implicit operation. When a multicast receiving context closes, XTP should notify XTP GM.

Delete(*XTP_group_id, access*) → {*result, membership*}

This primitive deletes a multicast group. The local XTP GM broadcasts to all remote XTP GM entities requesting permission to delete the group; if any XTP GM entity has one or more local members of the group, it reports its group members. In this case Delete() fails. If no members are detected, the XTP GM name server is contacted to deallocate the group identifier. *Membership* is an integer value representing the number of members known to be left in the group.

On most networks, the name space for group identifiers should be orders of magnitude greater than the number of multicast groups, even taken over a lengthy time period. Under these circumstances, Delete() may rarely be used.

Modify_Cache(*XTP_group_id, attributes, attribute-values, access*) → {*result*}

XTP GM is a supporting mechanism for process level group management. Process level management schemes may acquire information about a multicast group's attributes, including membership, that has not been learned by the local XTP GM entity. The Modify_Cache() primitive allows applications to convey their knowledge of multicast groups to the XTP GM database. *Attributes* is a vector of the group's attributes to be modified; *attribute-values* is a vector of the new values.

Query_Cache(*XTP_group_id, attributes, access, local*) → {*result, attribute_values*}

This primitive allows the user to query XTP GM for the current value of the *attributes* of the group identified by *XTP_group_id*. The values are returned in *attribute_values* if the operation is successful.

If *local* indicates a local inquiry, XTP GM returns the information available in the local group management database. Otherwise, the local XTP GM entity queries remote sites, via a broadcast, and waits for some pre-defined time for replies. All XTP GM entities at remote sites on which members of the group identified by *XTP_group_id* reside will respond with their local view of the group. The XTP GM entity that initiated the query coalesces the responses, updates its local database, and returns the newly updated attributes to the caller.

A wildcard value of *attributes* indicates a request for all (externally visible) attributes of the group.

3. Reliable Multicast Primitive

MultiSend(*XTP_group_id, access, n-members, specific-members, message-buffers*) → {*result, success-list*}

The MultiSend() primitive provides reliable multicast transmission in which the user can specify reliable delivery in terms of (1) the total number of receivers that must be reached (*n-members*) and (2) a list of particular group members identified by their *XTP_member_ids* (*specific-members*) that must be among the set of successful receivers. These two parameters allow a powerful, flexible interface for covering the spectrum from cheap, less reliable service to expensive, highly reliable service. The application possesses a body of knowledge—application semantics such error detection and recovery and latency requirements, the relative

accuracy on average of group membership information supplied by process level group management and XTP GM, and possibly knowledge of system parameters or external factors that will affect the communication— that allows the application to select the most appropriate transfer semantics for a multicast.

In a call to MultiSend(), XTP GM builds a *transfer list* based on *n-members* and *specific-members*, submits it to XTP, and initiates an XTP reliable multicast. When message transmission completes, XTP GM returns a boolean value *result* and a list of group members known (by the XTP multicast context) to have received the distribution, *success-list*. The latter is not required to be a complete list of members that received the message. Nonetheless, this postmortem may sometimes be valuable and the client process is offered the opportunity to view it.

4. Implications for XTP

The proposed Group Management scheme has a number of implications for the XTP multicast facility. Under this scheme XTP CNTL packets from multicast receivers must carry the group member identifiers, *XTP_member_ids*, since these identifiers are used in the transfer list. In the strongest semantics for supporting the transfer list, an XTP multicast transmit context would release transmit buffers only after determining that the required receivers had reported reception of the data. A weaker form of support would forego use of the transfer list in error control algorithms, checking the source of CNTL packets only at the beginning (connection set-up) and end of multicast message delivery.

The interface between XTP multicast logic and the transfer list will play a crucial part in any implementation of the XTP GM facility. One scenario would be to have a 'side buffer' for the in-coming CNTL packets. The transmit context sets the WTIMER upon issuing a SREQ or DREQ. Upon expiration of the timer, all collected CNTL packets are processed using the

transfer list, and the context state updated. If no progress is made after some number of iterations of this process, the connection is broken. In a hardware implementation, CNTL packet processing (e.g. doing look-ups into the transfer list and coalescing the state of the multiple receivers in the transfer list into a directive for actions within the transmitting context) could take place in parallel with other protocol processing.

In a transfer list-driven reliable multicast, the transmitter must be able to collect control information from the receiver set effectively. The XTP 3.4 Definition indicates that *damping* is applied to CNTL packets issued in response to DREQ and (presumably) SREQ bits in DATA or FIRST packets. Hence, as the size of the multicast group increases, damping may begin to adversely affect the ability of the multicast transmitter to obtain control information from certain receivers by setting the DREQ or SREQ bit in out-going packets. The multicast transmitter can not detect the difference between a receiver that is consistently damped and a failed receiver.

XTP GM's control over naming multicast group members enables the possibility of including a simple mechanism in XTP for improving the ability of a multicast sender to solicit CNTL packets from lagging or silent receivers. *XTP_member_ids* could contain a *subgroup* field of, for example, 8 bits, which indicated the subgroup to which a group member belonged. A member's subgroup would be randomly assigned with 8 distinct subgroups, one for each bit position. In every DATA and FIRST packet issued from the multicast sender, an 8-bit field would be carried indicating the subgroups that should respond to the SREQ header bit. Upon receiving a DATA packet with the SREQ bit set, a multicast receiver would check to see if its subgroup bit was set in the DATA packet. If not, a CNTL packet would not be issued.